Robust Coding and Debugging

CSE 403

Outline

- Writing solid code
- Errors, asserts, exceptions
- Debugging practices

Resources

- 403 Sp'05
- "The Practice of Programming," by Brian Kernighan and Rob Pike
- "The Pragmatic Programmer," by Andrew Hunt and David Thomas

"It's a painful thing
To look at your own trouble and know
That you yourself and no one else has made it."
– Sophocles, Ajax

Don't do this

```c
#include <stdio.h>

void FreeMemory(void *pv) {
    Assert(pv != NULL);
    memset(pv, 0xA3, sizeof(*pv));
    free(pv);
}
```

Writing solid code

- Shred your garbage
- Force early failure, increase determinism
- Why 0xA3?
Coding quiz

```c
char tolower(char ch)
{
    return ch - 'A' + 'a';
}
}
```

Handling out of range inputs

- Ignore
- Return error code
- Assert
- Redefine the function to do something reasonable
- Write functions that, given valid inputs, cannot fail

Candy machine interfaces

- Error prone return values or arguments
  ```c
c char c;
c = getchar();
if (c == EOF) ...
```
  - Classic bad example, getchar() returns an int!
- Alternate approach
  ```c
bool fGetChar(char pch);
```
- Many bugs with malloc returning NULL

Assertions

- Don’t use assertions to check unusual conditions
  - You need explicit error code for this
  - Only use them to ensure that illegal conditions are avoided

Exceptions

- Put error handling in a single place
- Exceptions should be reserved for unexpected events
  - It is exceptional if a file *should* be there but isn’t
  - It is not exceptional if you have no idea if the file should be exist or not

Debugging

- What are the key steps in debugging a program?
Step through your code

- Maguire
  - Step through new code in the debugger the first time it is used
    - Add code, set break points, run debugger
    - Add code, run tests, if failure, run debugger
- Knuth
  - Developed tool to print out first two executions of every line of code

Kernigan and Pike's debugging wisdom

- Look for common patterns
  - Common bugs have distinct signatures
    - int n; scanf("%d", &n);
- Examine most recent change
- Don't make the same mistake twice
- Debug it now, not later
- Get a stack trace
- Read before typing

K & P, II

- Explain your code to someone else
- Make the bug reproducible
- Divide and conquer
  - Find simplest failing case
- Display output to localize your search
  - Debugging with printf()
- Write self checking code
- Keep records

Don't do this

```java
try {
    doSomething();
} catch (Exception e) {
    // ...
}
```

- Can cover up very bad things
- Violates K&P: Debug it now, not later

Should debug code be left in shipped version

- Pro:
  - Debug code useful for maintenance
  - Removing debug code change behavior
    - Bugs in release but not debug versions
- Con:
  - Efficiency issues
  - Different behavior for debug vs. release
    - Early fail vs. recover

Apocryphal (but still a good story)

- A program which fails only in the month of September
Apocryphal (but still a good story)

- A program which fails only in the month of September

```c
char monthName[9];
strcpy(monthName, "September");
```